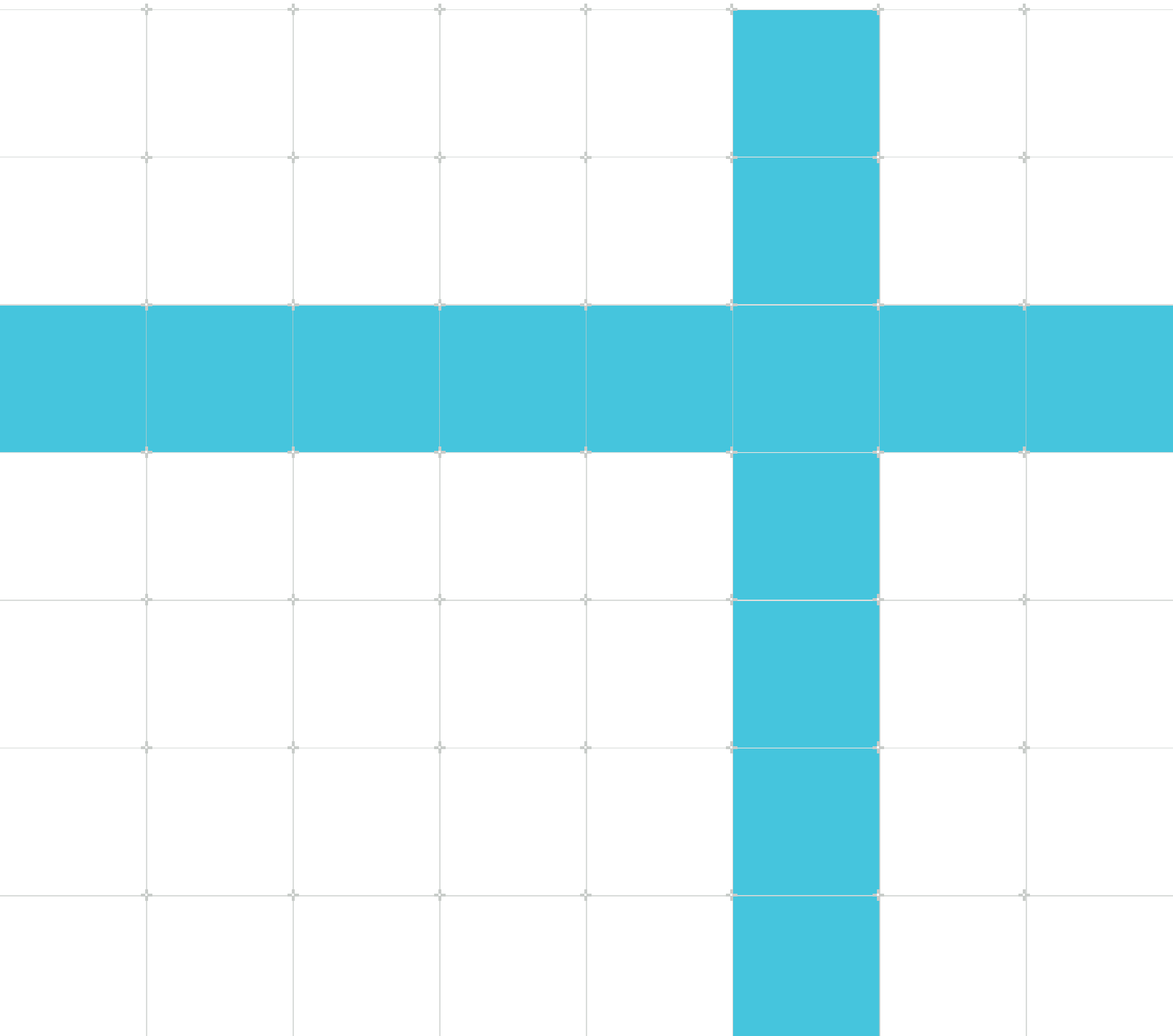
On-Target Trace Using the CoreSight Access Library

Version 1.0

Issue 02
102713_0100_02_en



On-Target Trace Using the CoreSight Access Library

Release information

Document history

Issue	Date	Confidentiality	Change
0100-02	4 November 2019	Non-Confidential	First release

Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly

or indirectly, in violation of such export laws. Use of the word "partner" in reference to Arm's customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349|version 21.0)

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is Final, that is for a developed product.

Feedback

Arm welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Contents

1. On-Target Trace and Profiling

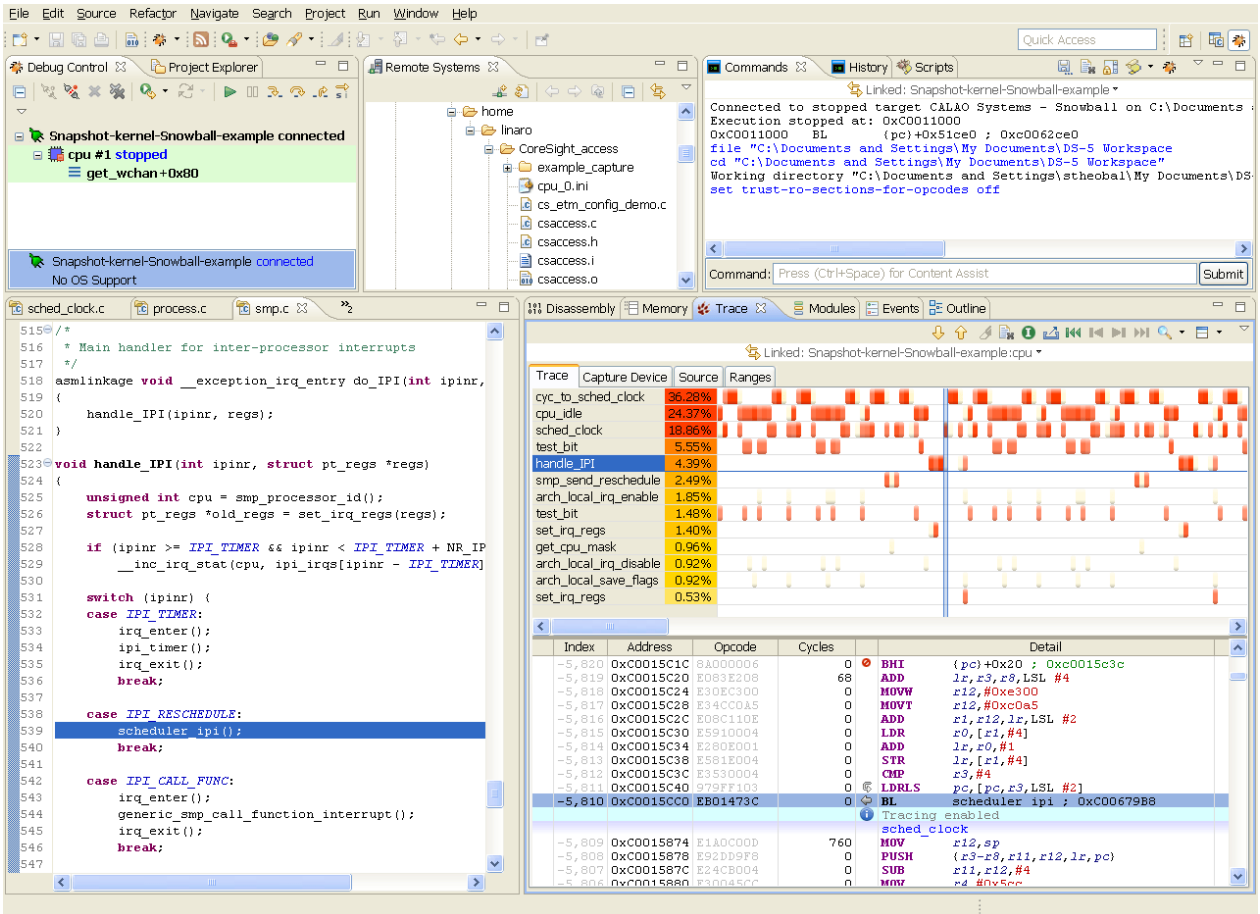
In this tutorial, we are using the CoreSight Access Library on a Cortex-A9-based ST-Ericsson Snowball board designed by Calao Systems, running Linux Ubuntu, to get a trace of instruction execution and to profile the usage of functions within the Linux kernel itself. The user-space example could be modified for real-time “flight-recorder” monitoring, or for post-mortem crash analysis.

Non-Intrusive Tracing and Profiling of Linux Kernel using the CoreSight Access Library

CoreSight Trace enables you to non-intrusively collect the sequence of instructions that were executed on the target platform - which is really useful when trying to debug thorny real-time issues, or when trying to optimize your code.

Here is a screenshot showing the result of a trace capture - you can see the actual sequence of instructions executed in the kernel, the corresponding function in the C source file, and which functions occupied the most time.

Figure 1-1: Trace capture result



In the navigational timeline, the colour coding is a heat map showing the executed instructions and the amount of instructions each function executes in each timeline - the darker red showing more instructions and the lighter yellow showing fewer instructions.

At a scale of 1:1 the colour scheme changes to display memory access instructions as a darker red colour, branch instructions as a medium orange colour, and all the other instructions as a lighter green colour.

2. What can CoreSight trace do?

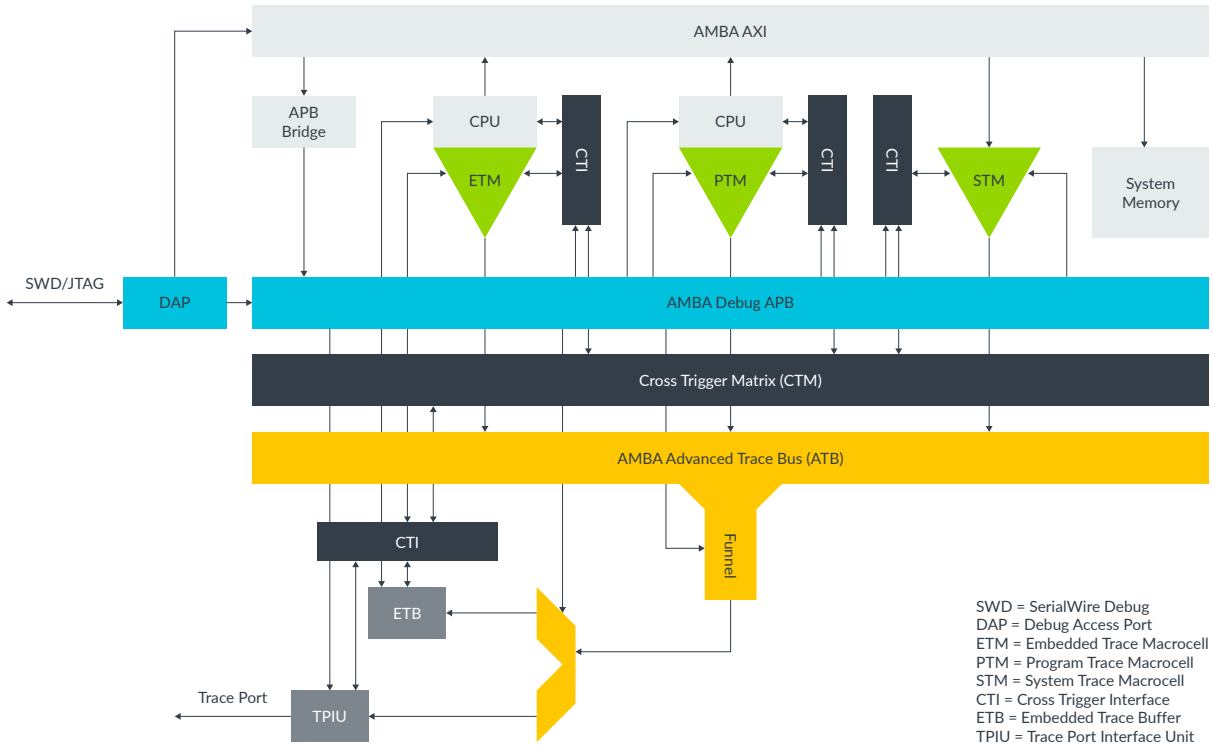
Trace is available on a variety of Arm processors, for example, the Cortex-A9 processor core can provide a trace interface to an (optional) CoreSight Program Trace Macrocell (PTM) that allows tracing of program flow (instructions only, not data).

The PTM outputs the raw trace information to an on-chip Embedded Trace Buffer (ETB) or in more recent devices to an on-chip Embedded Trace FIFO (ETF), or to a parallel trace port via the Trace Port Interface Unit (TPIU) which can be connected to external trace-capable tools.

3. Example CoreSight System

The following diagram shows an example CoreSight system:

Figure 3-1: An example CoreSight system



Trace data captured in ETB/ETF can be read via JTAG using probes such as the Arm DSTREAM and Keil ULINK probes. Alternatively, trace data can be collected and stored by user code on the target for retrieval later, without the need for any JTAG probe.

Debuggers can retrieve the captured data to display a complete trace execution history, including branches, exceptions, and whether conditional instructions were executed or skipped. Timestamps, Context IDs and VMIDs can also be captured on higher-end devices.

The debugger can then match the instructions executed to the source code via ELF/DWARF debug information, to show which functions were being executed over time - code profiling. And because CoreSight trace is completely non-intrusive, the code runs at full speed on the target - capturing trace does not alter any timing aspects of the code, unlike other debug methods such as inserting `printf()` statements.

4. CoreSight Access Library

The CoreSight Access Library provided with DS-5 enables user code to interact directly with CoreSight devices on your target. This allows, for example, program execution trace to be captured in a production system without the need to have an external debugger connected.

The trace can be saved to a file on the target at run-time, and the saved trace can be retrieved later and loaded into DS-5 Debugger for analysis.

The library supports a number of different CoreSight components on several target boards - check the readme to see the current list. You can modify it to support other CoreSight components or other boards. The library can be used in a bare-metal or Linux environment.

The library offers a flexible programming interface allowing a variety of use cases and experimentation. It also offers some advantages compared to a register-level interface. For example, it can:

- Manage any unlocking and locking of CoreSight devices via the lock register, OS Lock register, programming bit, power-down bit.
- Attempt to ensure that the devices are programmed correctly and in a suitable sequence.
- Handle variations between devices (for example, between variants of ETM/PTMs), and where necessary, work around known issues.
- Become aware of the trace bus topology and will generally manage trace links automatically. For example enabling only funnel ports in use.
- Manage “claim bits” that coordinate internal and external use of CoreSight devices.

5. Using the CoreSight Access Library Example

An example Linux application, called tracedemo is provided in DS-5 that uses the CoreSight Access Library. As it runs, tracedemo creates several files on the target, including the captured trace. After running, the saved files can copied from the Linux target to your host computer, either using a secure copy program such as scp (Windows versions of this Linux command are available, such as pscp as provided with PuTTY), or using Remote System Explorer as provided in DS-5 Debugger.

6. Importing the example

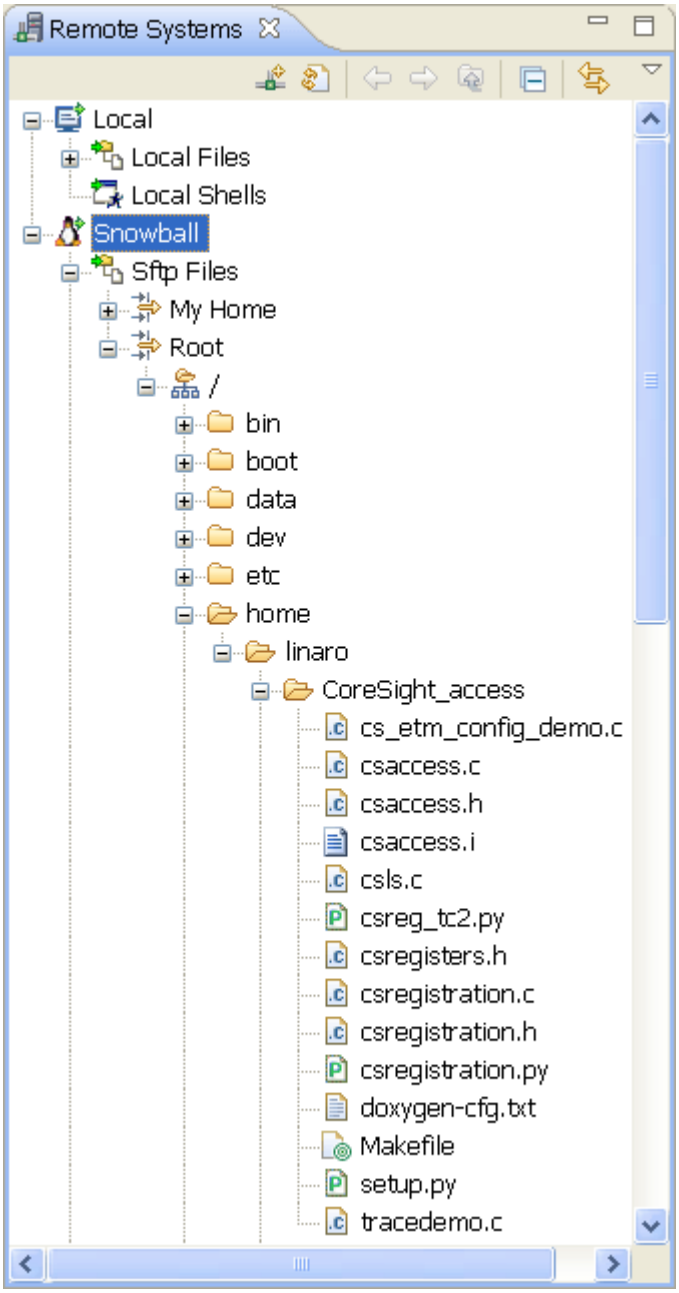
The sources that are released with DS-5 are provided in: `[DS-5 install dir]\examples\CoreSight_Access_Library.zip.`

If you require the latest sources, you can download it from the CoreSight Access Library.

First, import the example into the DS-5 Eclipse workspace as follows:

1. Launch Eclipse for DS-5
2. Close the Welcome screen, if it appears
3. Go to File menu and select Import.... The Import wizard opens
4. Open the General folder
5. Select Existing Projects into Workspace
6. Press Next
7. Press Browse... next to Select archive file:. A file dialog opens
8. Navigate to `[DS-5 install dir]\examples, select Coresight_Access_Library.zip`, then press OK. The Projects: field populates with the project(s) found in the .zip
9. Press Finish. The Project view populates with the project(s)

Figure 6-1: Remote systems overview



Next, boot Linux on your target and login (preferably as root). Then copy the `CoreSight_access` directory from the workspace to a suitable writable directory on your Linux target, either using Remote System Explorer as provided in DS-5 Debugger or using a secure copy program such as scp (Windows versions of this Linux command are available, such as `pscp` as provided with PuTTY). You'll need to get the IP address of the target to do this, using `ipconfig`.

7. Configuring the example for your Target

Before you can use the example on your target, you may have to modify some of the example source files to suit your target:

- If your target is not one of the platforms already supported, then you will need to add a registration function into `csregistration.c`
- To configure the range of addresses to be traced in the kernel, open `tracedemo.c` and modify the values of:
  - `KERNEL_TRACE_SIZE` is the extent of the region to trace
  - `KERNEL_TRACE_VIRTUAL_ADDR` is the virtual address of the start of the region to trace, i.e. corresponding to addresses in the vmlinux file
  - `KERNEL_TRACE_PHYSICAL_ADDR` is the physical address (e.g. in RAM) of the start of the region to trace, normally a fixed offset from `KERNEL_TRACE_VIRTUAL_ADDR`

One way to identify the addresses of candidate regions/functions to trace is to use, for example:

```
grep cpu_idle /proc/kallsyms
```

8. Configuring your Target for the example

Before using the example on your target, check the following:

- For tracedemo to use `mmap()` to read kernel memory (which it then dumps to a file `kernel_dump.bin` that can be read by DS-5 Debugger), the kernel must have been built without `"CONFIG_STRICT_DEVMEM=y"`. You can check this by typing: `zgrep "CONFIG_STRICT_DEVMEM" /proc/config.gz` If the kernel has been built with `"CONFIG_STRICT_DEVMEM=y"`, then you must either rebuild it with `"CONFIG_STRICT_DEVMEM=n"`, or dump the kernel memory manually by some other means, for example, use DS-5 Debugger and DSTREAM to dump this kernel memory from the target (with, for example, the CLI command `"dump memory kernel_dump.bin 0xC000D000 +0x5000"`), or extract this kernel memory from the kernel Image file.
- To view the disassembly and trace augmented with symbols (e.g. function names), the vmlinux file for the kernel must be available, built with debug symbols. To map the trace and disassembly back to kernel source files, the kernel source tree must also be available. The vmlinux file with debug symbols is created when the kernel is built with debug info `"CONFIG_DEBUG_INFO=y"`. You can check how your kernel was built by typing: `zgrep "CONFIG_DEBUG_INFO" /proc/config.gz` If using `make menuconfig` to rebuild the kernel, select the following under Kernel Hacking:

```
[*] Kernel debugging
      [*] Compile the kernel with debug info
```

9. Building the example and library

The supplied makefile expects the example and library to be built natively on the target; minor modifications will be needed if they are built using a cross-compiler. The example and library can be built by typing:

```
make tracedemo
```

You should see, for example:

```
gcc -O2 -Wall -Wno-switch -g -o tracedemo.o -c tracedemo.c
gcc -O2 -Wall -Wno-switch -g -o csaccess.o -c csaccess.c
ar cr libcsaccess.a csaccess.o
gcc -O2 -Wall -Wno-switch -g -o csregister.o -c csregistration.c
gcc -o tracedemo tracedemo.o libcsaccess.a csregister.o
```

10. Building the API documentation

To create the API documentation, you need to have installed Doxygen. Then run:

```
make docs
```

The resulting API documentation can then be found in doc/html.

11. Running the example

Run the tracedemo Linux application as root with:

```
./tracedemo
```

This will auto-detect the target board (if supported), configure trace generation, and retrieve trace from the trace buffer.

To view the possible run options, add `-h` for help:

```
./tracedemo -h
```

You will see output like:

```
Usage:
-h              This help screen.
-c <cpu>        Select CPU for demo to run on. Default 0
-itm            Enable ITM tracing, ITM tracing disabled by default.
-full           Show full trace with no filtering.
-pause          Run the demo with a pause after each step.
```

If you see:

```
** csaccess: ERROR: can't open /dev/mem
```

then try running it as sudo:

```
sudo ./tracedemo
```

You will see output like this sample captured from a Snowball board.

Depending on the configuration, tracedemo creates a number of output files such as `snapshot.ini`, `cpu_0.ini`, `device_0.ini`, `device_1.ini`, `trace.ini`, `kernel_dump.bin` and `cstrace.bin`. These files can be loaded into DS-5 Debugger to analyze the collected trace, as described in the next section. Copy these files from the target into a suitable writable directory on your host, either using Remote System Explorer or scp (or pscp).

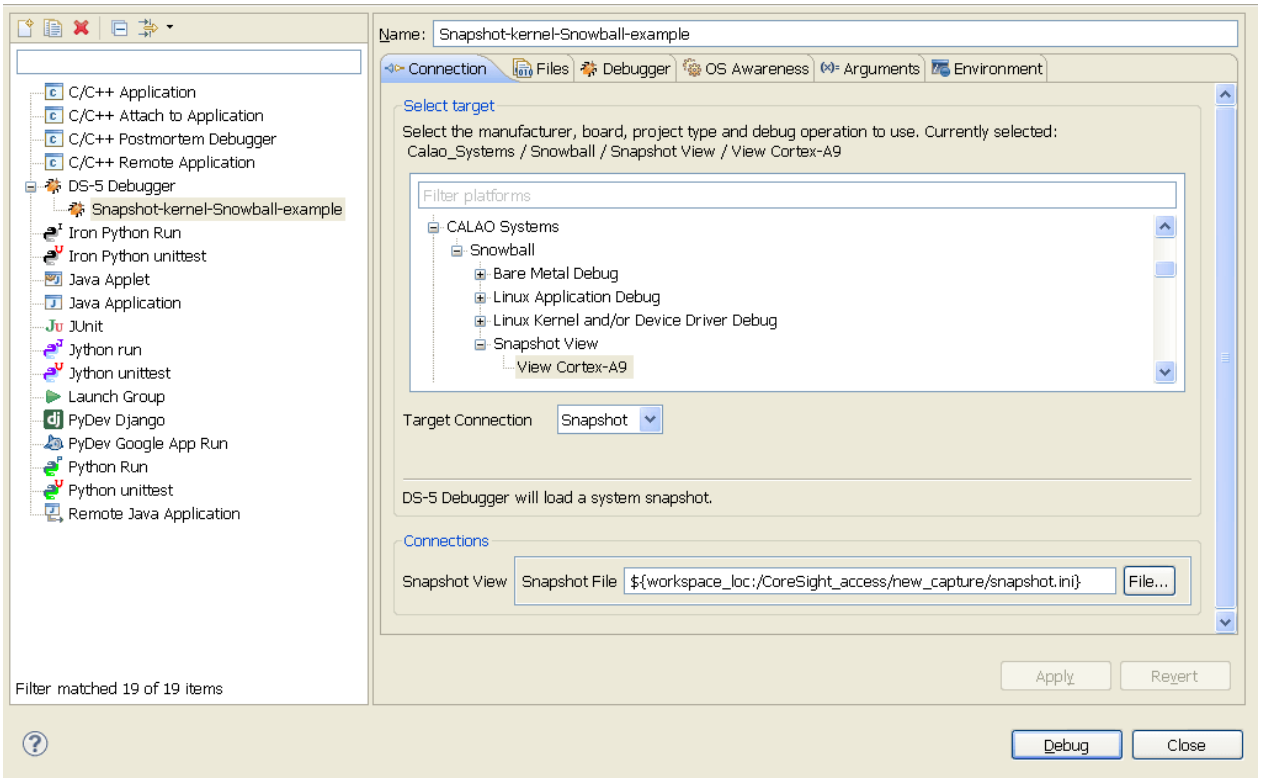
To create a standalone package of files for decoding by DS-5 Debugger, for example to give to someone else, collect the files above, together with vmlinux, and the kernel source tree.

12. Analyzing the collected trace in DS-5 Debugger

The files created by tracedemo, after copying to the host, can be loaded into DS-5 Debugger to analyze the collected trace as follows. Ready-made example files captured from a session on a Snowball target are provided in the `\example_capture` directory that can be used for test or demo purposes. A ready-made debug config launch file Snapshot-kernel-Snowball-example is also provided.

1. Launch Eclipse for DS-5.
2. Go to Window menu and select Open Perspective > DS-5 Debug.
3. Go to Run menu and select Debug Configurations…. The Debug Configurations dialog opens.
4. Open the DS-5 Debugger node.
5. Create a new debug configuration and give it a name.
6. In the Connection tab tree view, select your target and the Snapshot View entry for it.
7. In the Connection field below, enter the path to the top-level contents file `snapshot.ini`. You can use the File… button to navigate to the file. See the screenshot below. To load the ready-made example files, specify e.g. `...\DS-5 Workspace\CoreSight_access\example_capture \snapshot.ini`.
8. To view the disassembly and trace augmented with symbols (e.g. function names), the vmlinux file for the kernel must be given to DS-5 Debugger. In the Files tab, select Load symbols from file, and use the File System… or Workspace… buttons to navigate to the vmlinux file.
9. In cases where the kernel modifies itself, you can force DS-5 Debugger to use program code from `kernel_dump.bin` rather than vmlinux. In the Debugger tab, tick Execute debugger commands, and add set trust-ro-sections-for-opcodes off to its field. 10 Press Debug.

Figure 12-1: Debug launcher



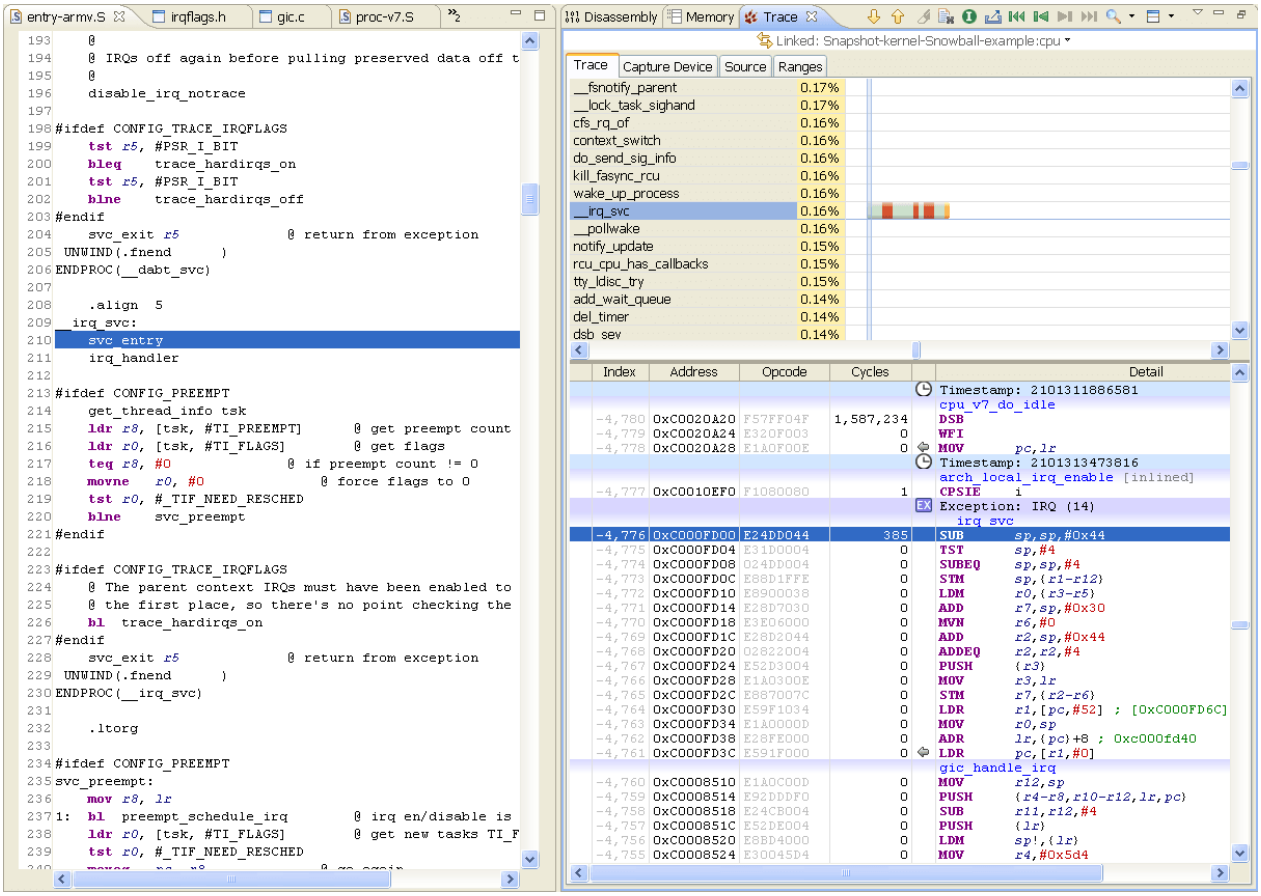
13. Digging deeper into the trace

The trace captured earlier was focussed in a narrow range of addresses around `cpu_idle()`. By widening the trace range, events such as IRQ exceptions can also be captured, so that the execution of the interrupt handler can analyzed.

As these events are relatively rare, DS-5 Debugger has configurable filters so that, for example, only exceptions are displayed, so that these can be spotted more easily. DS-5 Debugger is also able to export the captured trace data into a text file.

In the following screenshot, we can see an IRQ exception occurring immediately after interrupts are re-enabled by a `CPSIE` instruction within `arch_local_irq_enable()` at index 4,777. The interrupt is then handled by the code in `__irq_svc()`.

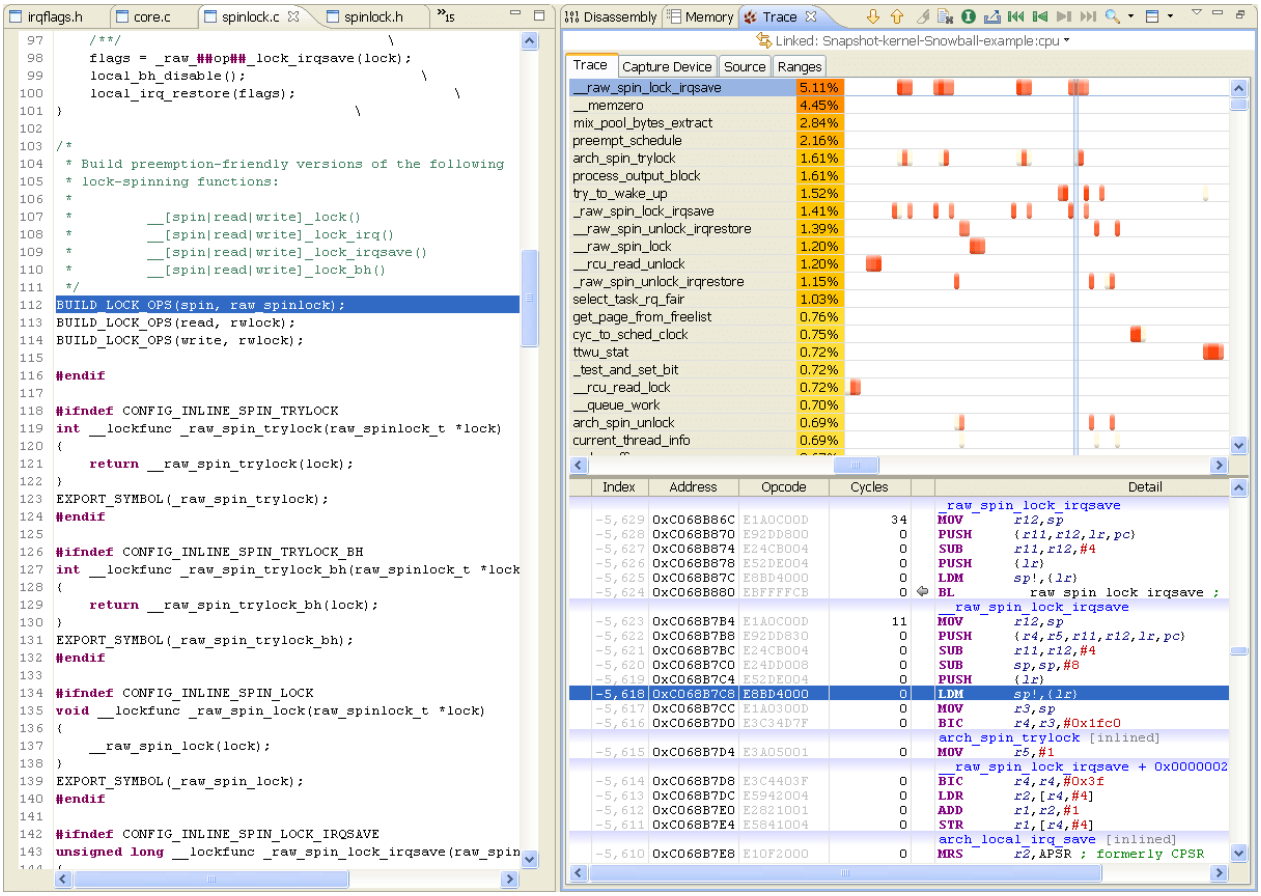
Figure 13-1: Tracing through the IRQ handler in the Linux Kernel



14. Profiling the Kernel

In the following screenshot, we can see the profile of a quiescent system, and that two functions together `__raw_spin_lock_irqsave()` and `__memzero()` occupy nearly 10% of the run-time. The red hot-spots in the heat-map reveal the instructions that occupy most time (mostly memory access instructions in this system). This reveals which functions are candidates for closer examination and possible optimization.

Figure 14-1: Profiling of a quiescent system



15. Summary

We've seen how the CoreSight Access Library can be used to get a trace of instruction execution within the Linux kernel non-intrusively, and how DS-5 Debugger's Snapshot Viewer can allow us to dig deep into the trace to debug the kernel, or to profile the usage of functions within the kernel. The user-space application running on the Linux target could be modified for real-time flight-recorder monitoring, or for post-mortem crash analysis.